On Demand secure routing protocol resilient to Byzantine failures

Primary Reference: B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, “An on-demand secure routing protocol resilient to Byzantine failures,” in *ACM Workshop on Wireless Security (WiSe)*, September 2002.

Student Lecture by
Ragavendra Ananthapadmanabhan
EECS 800 Survivable Networking

Department of Electrical Engineering and Computer Science
University of Kansas

OUTLINE

- INTRODUCTION
- AD HOC ROUTING VULNERABILITIES
- PROTOCOL PHASES OVERVIEW
- PROBLEM DEFINITION AND MODEL
- SECURE ROUTING PROTOCOL
  - Route discovery with fault avoidance
  - Byzantine fault detection
  - Link Weight management
- CONCLUSION AND FUTURE WORK

INTRODUCTION

- Ad Hoc Networks
  - No Fixed infrastructure
  - Highly suitable for emergency deployments
  - Search and rescue missions
  - High mobility and constrained power resources
  - Routing must converge quickly
- Traditional versus On demand routing
  - Distance vector and Link state consume power
  - Not designed to track topology changes at high rate
  - On Demand routing – Route request on demand
  - Routes can be cached till the topology changes

AD HOC ROUTING VULNERABILITES

- Common routing Vulnerabilities
  - Malicious nodes advertising false information
  - Try to redirect routes
  - Perform denial of service attack
  - More vulnerable to attack due to co-operative nature
- Byzantine Attacks
  - Attacker may have full control of the network
  - Performs *Black Hole* attacks
  - Create routing loops and high delay paths
- Protocol Assumptions
  - Attribute the fault to the link rather than the nodes
  - A path exists if there is a fault free link between the nodes
  - Provide routing survivability under the adversarial model

PROTOCOL PHASES OVERVIEW

- Route discovery with fault avoidance
  - Using flooding and a fault link weight list, find a least weight path
- Byzantine fault detection
  - Finds faulty links from source to destination
  - Adaptive probing technique is used
  - Faulty link is identified after log n faults
  - n is the length of the path
- Link weight management
  - Maintains a weight list of links
  - Faulty links are rehabilitated over time
  - Serves as input to the Route discovery phase

PROBLEM DEIFINITION AND MODEL

- Network Model
  - Requires bi-directional link between nodes
  - Do not address the attacks against lower layers
- Security model and considered attacks
  - Only source and destination are trusted
  - Intermediate nodes can be authenticated
  - Intermediate nodes may exhibit Byzantine behavior
- Byzantine Behavior
  - A node's behavior which causes disruption/degradation
  - Generally referred to as faults
- Problem Definition
  - Robust on demand routing protocol
  - Operating under the models above

SECURE ROUTING PROTOCOL

- On Demand protocol functionalities
  - Establishes reliability metric based on past history
  - High weights indicate low reliability
  - Each node maintains a weight list, updates it dynamically
  - Faulty links are identified using adaptive probing technique
  - Faulty links are avoided using the fault detection phase
- Phases block diagram

Figure 1: Secure Routing Protocol Phases

SECURE ROUTING PROTOCOL

- Route Discovery with fault avoidance
  - Flood the route request to guarantee delivery
  - Flood the response to prevent adversary from blocking the path
  - If blocked, then the route discovery will not have an input
  - No route caching is used due to security implications
  - Chooses a minimum weight path from the knowledge of LWM
  - Digital signing at intermediate nodes provides security
- Digital signatures
  - Authenticates the source sending requests
  - An unauthorized request will fail and will be dropped
  - Provides secure and legitimate route request

SECURE ROUTING PROTOCOL

- Request Initiation
  - Digitally signed request is broadcasted
- Request Propagation
  - Propagated through flooding at intermediate nodes
  - Verify matching request, source’s sign and rebroadcast
- Request Receipt/Response Initiation
  - The destination verifies the request, creates response
  - Response includes sequence number and weight list
- Response propagation/receipt
  - Compute the total weight by summing the individual weights
  - If less than the previous request, verify and broadcast
  - If the path in the response is better, update the route

SECURE ROUTING PROTOCOL

- Code to perform route discovery operation

Code executed at node source **when a new route to node** destination **is needed:**

```
(1)  CreateSignSend( REQUEST, destination, source, req_sequence, weight_list )
```

Code executed at node this_node **when a request message** req **is received:**

```
(2)  if( Find( requests_list, req) = NULL )
(3)      VerifySignature( req.source, req.signature )
(4)      if( this_node = req.destination )
(5)          CreateSignSend( RESPONSE, req.destination, req.source, req.req _sequence, req.high_weights_list )
(6)      else
(7)          Broadcast( req )
(8)      endif
(9)      InsertList(requests_list, req)
(10) endif
```

SECURE ROUTING PROTOCOL

- Code to perform route discovery operation

Request initiation

Code executed at node source when a new route to node destination is needed:

```
(1)  CreateSignSend( REQUEST, destination, source, req_sequence, weight_list )
```

Code executed at node this_node when a request message req is received:

```
(2)  if( Find( requests_list, req) = NULL )
(3)      VerifySignature( req.source, req.signature )
(4)      if( this_node = req.destination )
(5)          CreateSignSend( RESPONSE, req.destination, req.source, req.req _sequence, req.high_weights_list )
(6)      else
(7)          Broadcast( req )
(8)      endif
(9)      InsertList(requests_list, req)
(10) endif
```

SECURE ROUTING PROTOCOL

- Code to perform route discovery operation

Request initiation

```
Code executed at node source when a new route to node destination is needed:
(1)  CreateSignSend( REQUEST, destination, source, req_sequence, weight_list )

Code executed at node this_node when a request message req is received:
(2)  if( Find( requests_list, req) = NULL )
(3)      VerifySignature( req.source, req.signature )
(4)      if( this_node = req.destination )
(5)          CreateSignSend( RESPONSE, req.destination, req.source, req.req _sequence, req.high_weights_list )
(6)      else
(7)          Broadcast( req )
(8)      endif
(9)      InsertList(requests_list, req)
(10) endif
```

Indicates no matching requests

SECURE ROUTING PROTOCOL

```
Code executed at node this_node when a response message res is received:
(11) update = false
(12) prev_node = res.destination
(13) total_weight = 0
(14) for( i = 0; i < res.no_hops; i++ )
(15)     total_weight += LinkWeight( res.weight_list, prev_node, res.hops[i].node )
(16)     prev_node = res.hops[i].node
(17) endfor
(18) res.total_weight = total_weight + LinkWeight( res.weight_list, prev_node, this_node )
(19) prev_response = Find( responses_list, res )
(20) if( prev_response ≠ NULL )
(21)     if( res.total_weight ≥ prev_response.total_weight )
(22)         update = true
(23)     endif
(24) else
(25)     update = true
(26) endif
(27) if( update )
(28)     VerifySignature( res.destination, res.signature )
(29)     for( i = 0; i < res.no_hops; i++ )
(30)         VerifySignature( res.hops[i].node, res.hops[i].signature )
(31)     endfor
(32)     if( this_node = source )
(33)         UpdateList( path_list, res )
(34)     else
(35)         CreateSignSend( res, this_node )
(36)         UpdateList( responses_list, res )
(37)     endif
(38) endif
```

SECURE ROUTING PROTOCOL

```
Code executed at node this_node when a response message res is received:
(11) update = false
(12) prev_node = res.destination
(13) total_weight = 0
(14) for( i = 0; i < res.no_hops; i++ )
(15)     total_weight += LinkWeight( res.weight_list, prev_node, res.hops[i].node )
(16)     prev_node = res.hops[i].node
(17) endfor
(18) res.total_weight = total_weight + LinkWeight( res.weight_list, prev_node, this_node )
(19) prev_response = Find( responses_list, res )
(20) if( prev_response ≠ NULL )
(21)     if( res.total_weight ≥ prev_response.total_weight )
(22)         update = true
(23)     endif
(24) else
(25)     update = true
(26) endif
(27) if( update )
(28)     VerifySignature( res.destination, res.signature )
(29)     for( i = 0; i < res.no_hops; i++ )
(30)         VerifySignature( res.hops[i].node, res.hops[i].signature )
(31)     endfor
(32)     if( this_node = source )
(33)         UpdateList( path_list, res )
(34)     else
(35)         CreateSignSend( res, this_node )
(36)         UpdateList( responses_list, res )
(37)     endif
(38) endif
```

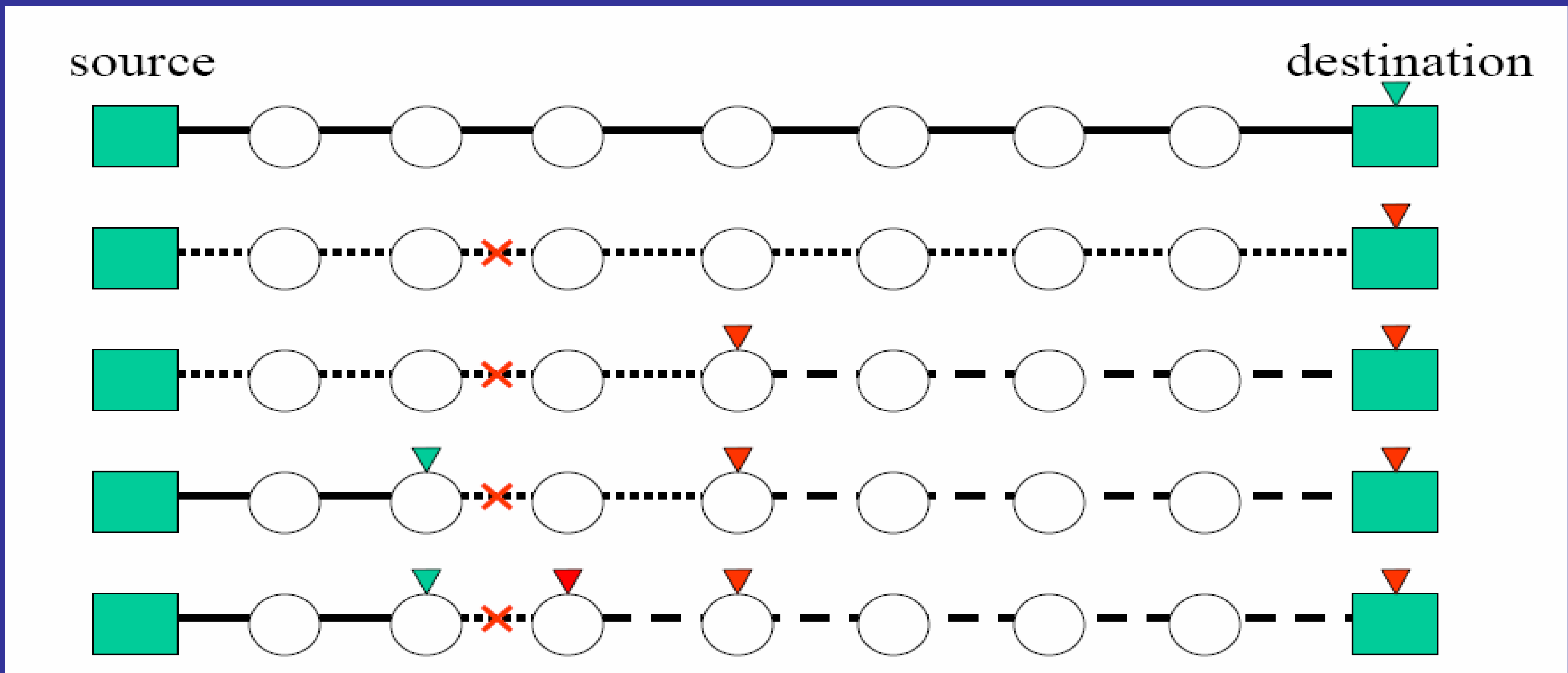
Total weight calculation

SECURE ROUTING PROTOCOL

- Byzantine Fault Detection
  - Detection algorithm uses ACKs with timeout
  - Packets not received before timeouts are considered as loss
  - Loss threshold is set which sets the limit of damage by attacker
  - Network performance is not affected due to identification of links
  - DSR Topology changes notification [2] allows optimization
  - A route error message is propagated to inform the change
- Fault Detection overview
  - Source keeps track of ACKs received
  - If no. of recent losses exceeds threshold, register a fault
  - Start a binary search on the path to identify faulty link

SECURE ROUTING PROTOCOL

- Fault detection overview
  - Probe nodes to search for faulty links



SECURE ROUTING PROTOCOL

- Probe specification
  - Specified in the list in the same order as they appear in path
  - Probe send ACK based on verifying HMAC and encryption
  - Verify the contents of the packets at all nodes
- Acknowledgement specification
  - ACKs from previous nodes are combined and sent
  - This ensures reception of ACKs from the nodes in the path
  - Thus the source discovers a loss on an interval
- Probe retirement – due to overhead
  - Counter is assigned for each joined probe
  - Successful ACK decreases the counter by one.
  - When Counter becomes zero, withdraw probe

SECURE ROUTING PROTOCOL

- Link Weight Management
  - Provides input to the route discovery process
  - Management scheme maintains the weight list
  - Weight list generated from the faulty link output of previous phase
  - When a link is faulty, its weight is doubled
  - Higher weight indicates lower reliability
  - Lower reliability routes will not comprise the path

CONCLUSION AND FUTURE WORK

- Successfully identifies faulty links
- Byzantine behavior is easily identified
- Survivability is provided even in presence of attacks
- Other Byzantine behaviors to be mitigated
- Threshold levels to be studied
- Route caching with security

REFERENCES

- [1] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, “An on-demand secure routing protocol resilient to Byzantine failures,” in *ACM Workshop on Wireless Security (WiSe)*, September 2002.

- [2] D.B. Johnson, D. A. Maltz, and J. Broch, DSR: The Dynamic source routing protocol for multi hop wireless ad hoc networks, in Ad Hoc networking, Ch. 5, pp. 139-172. Addison-Wesley, 2001.